

Indian Institute of Technology Madras
ZANZIBAR CAMPUS

Z5007 · PROGRAMMING AND DATA STRUCTURES

Measuring Algorithms Without Running Them: Asymptotic Analysis

---

Lecture 2 · Wednesday 7 October 2026 · Canning, Broder & Lafore, Ch 2 (Big O Notation)

Dr. Innocent Nyalala | MTech Semester 1 | IIT Madras Zanzibar

Today's Plan

IIT Madras Zanzibar

1. (0–15 min) Warm-up quiz: printed, ungraded
2. (15–17 min) Your expectation cards: what you told me
3. (17–22 min) Why timing a program is not enough
4. (22–28 min) Counting steps: the RAM model, worked examples
5. (28–37 min) Big-O, Big-Omega, Big-Theta: definitions and a proof
6. (37–43 min) Growth classes and three rules for loops
7. (43–50 min) Exercise, solution, summary

Learning Outcomes

IIT Madras Zanzibar

By the end of this session, you will be able to:

Explain	why running time in seconds is a poor way to compare algorithms
Count	the steps of a simple program as a function of the input size n
Prove	that a function is O(g(n)) by finding the constants c and n_o
Classify	loops and code fragments as O(1), O(log n), O(n), O(n log n) or $O(n^2)$

Measuring Algorithms Without Running Them

IIT Madras Zanzibar

→ **Warm-up Quiz**

Why Not Just Time It?

Counting Steps

Big-O, Big-Omega, Big-Theta

The Growth Classes and Loop Rules

In-Class Exercise

Warm-up Quiz: 15 Minutes, Not Graded

IIT Madras Zanzibar

15 minutes

not graded · name optional · 'don't know' is a good answer

A

Part A

About you: your background, Python comfort, prior courses

B

Part B

10 short questions: Python, stacks, queues, recursion, sorting, Big-O

Why: it tells me where each of you stands, so the next three weeks fit the room. Answer key on Google Classroom tonight.

Your Expectation Cards: Thank You

IIT Madras Zanzibar

1 What you expect

the three most common expectations, read aloud

2 What worries you

the three most common worries, and my answer to each

My promise: every worry on a card gets an answer, either today or in office hours (Tuesday 2–4 pm).

Measuring Algorithms Without Running Them

IIT Madras Zanzibar

- Warm-up Quiz
- → **Why Not Just Time It?**
- Counting Steps
- Big-O, Big-Omega, Big-Theta
- The Growth Classes and Loop Rules
- In-Class Exercise

Monday's Question: How Did I Know Before Running It?

IIT Madras Zanzibar

Monday's example	Strategy	Result	Growth
Duplicates, program A	compare every pair	about 6.5 hours	n^2
Duplicates, program B	sort, check neighbours	0.45 seconds	n log n
Duplicates, program C	set of seen IDs	0.19 seconds	n
Guess my number, count up	1, 2, 3, ...	up to 100 guesses	n
Guess my number, halve	guess the middle	at most 7 guesses	log n

Today's question: how do we predict the last column with a pencil, before running any code?

Why Not Just Time It With a Stopwatch?

IIT Madras Zanzibar

1

The machine

a laptop, a phone and a cloud server give different seconds for the same code

2

The language

the same algorithm in C can run many times faster than in pure Python

3

The input size

program A: instant on 100 IDs, hours on 1,000,000. One timing hides the growth

What we want instead: a measure of how the WORK GROWS as the input size n grows, independent of machine and language.

Measuring Algorithms Without Running Them

IIT Madras Zanzibar

Warm-up Quiz

Why Not Just Time It?

→ **Counting Steps**

Big-O, Big-Omega, Big-Theta

The Growth Classes and Loop Rules

In-Class Exercise

The RAM Model: Count Steps, Not Seconds

IIT Madras Zanzibar

One step each (cost 1)

```
x = 5
```
assignment

```
a + b, a * b
```
arithmetic

```
a < b, a == b
```
comparison

```
a[i]
```
read one list element by index

```
return x
```
return from a function

Not one step

A loop: the cost of its body × the number of times it runs

A function call: the cost of everything inside the function

Python built-ins like sorted(a) or x in a: NOT one step (Thursday)

Notation: T(n) = the number of steps on an input of size n.

Worked Example 1: Summing a List

IIT Madras Zanzibar

```
1  def total(a):
2      s = 0
3      for x in a:
4          s = s + x
5      return s
```

Line	Steps each time	Times run	Total
2: s = 0	1	1	1
4: s = s + x	2 (add, assign)	n	2n
5: return s	1	1	1
T(n)			**2n + 2**

Check, n = 3: `1 + 6 + 1 = 8 = 2(3) + 2`

Double n, double the work: this is linear growth. We will write it as O(n).

Worked Example 2: Comparing Every Pair (Program A)

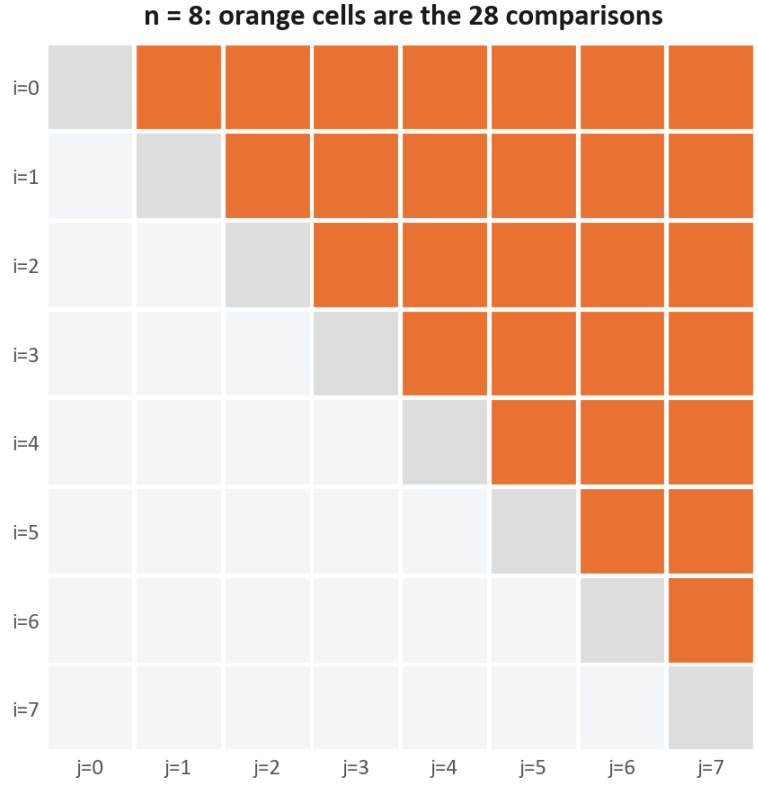
IIT Madras Zanzibar

```
def has_dup(a):
    n = len(a)
    for i in range(n):
        for j in range(i + 1, n):
            if a[i] == a[j]:
                return True
    return False
```

Comparisons: (n−1) + (n−2) + ... + 1

= n(n − 1) / 2

n = 8: 8 × 7 / 2 = 28



Double n, four times the work: n = 1,000,000 gives about 5×10^{11} comparisons. That is the 6.5 hours.

Only the Biggest Term Matters When n Is Large

IIT Madras Zanzibar

n	$n^2/2$	n/2	total	share from $n^2/2$
10	50	5	55	90.9%
100	5,000	50	5,050	99.0%
1,000	500,000	500	500,500	99.9%
10,000	50,000,000	5,000	50,005,000	99.99%

Rule 1: Drop lower-order terms

$n^2/2 + n/2 \rightarrow n^2/2$

Rule 2: Drop constant factors

$n^2/2 \rightarrow n^2$

Result: program A is $O(n^2)$. The constant 1/2 and the n/2 term do not change how the work grows.

Measuring Algorithms Without Running Them

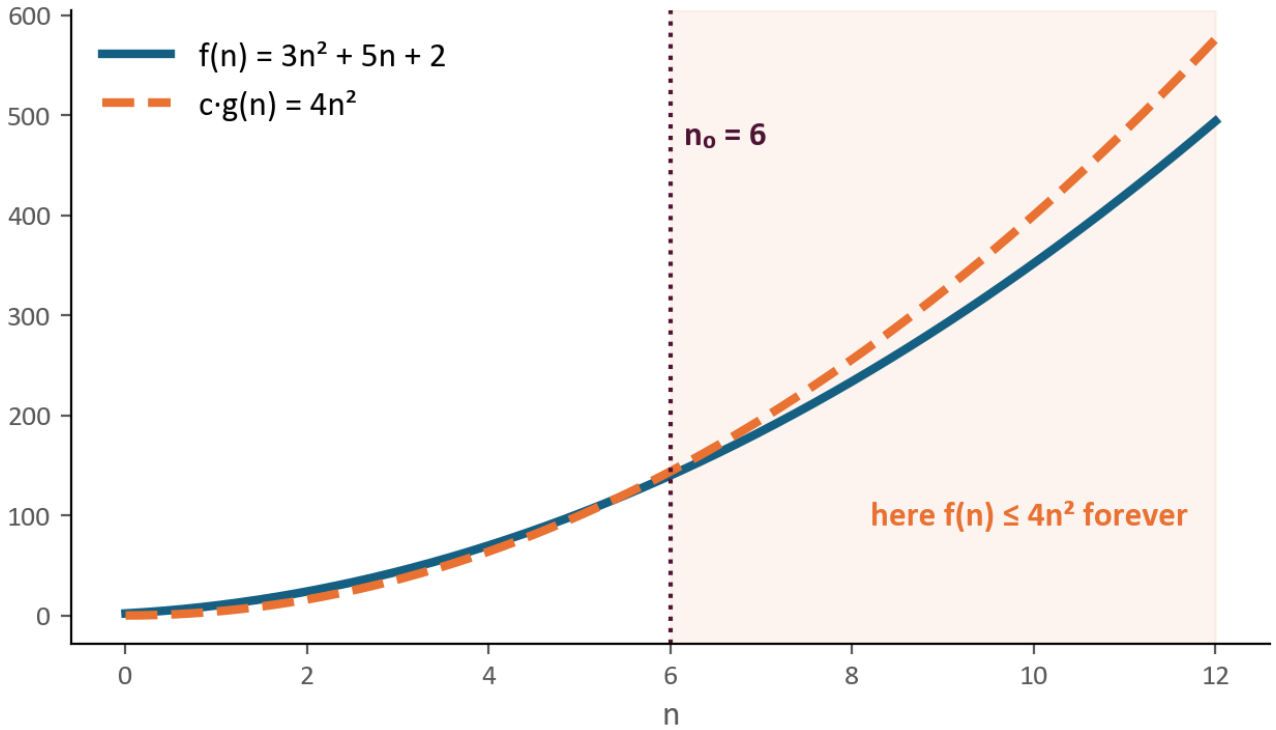
IIT Madras Zanzibar

- Warm-up Quiz
- Why Not Just Time It?
- Counting Steps
- → **Big-O, Big-Omega, Big-Theta**
- The Growth Classes and Loop Rules
- In-Class Exercise

Big-O: The Formal Definition

IIT Madras Zanzibar

$f(n) = O(g(n))$ if there are constants $c > 0$ and $n_0 \geq 1$ such that **$f(n) \leq c \cdot g(n)$ for all $n \geq n_0$**



In plain words

From some point n_0 on, f never rises above a constant multiple of g.

Big-O is an UPPER bound on growth.

Any c and n_0 that work are fine; they need not be the smallest.

Worked Proof: $3n^2 + 5n + 2$ Is $O(n^2)$

IIT Madras Zanzibar

1	**Goal**	find c and n_0 with $3n^2 + 5n + 2 \le c \cdot n^2$ for all $n \ge n_0$
2	**Trick**	for $n \ge 1$: $n \le n^2$ and $1 \le n^2$
3	**So**	$3n^2 + 5n + 2 \le 3n^2 + 5n^2 + 2n^2 = 10n^2$
4	**Done**	$c = 10$, $n_0 = 1$ works, so $3n^2 + 5n + 2 = O(n^2)$

Also valid: $c = 4$, $n_0 = 6$ (the chart). Any working pair, with reasoning, earns full marks.

Big-Omega and Big-Theta: Lower Bounds and Tight Bounds

IIT Madras Zanzibar

O

Big-O: upper bound

f grows no faster than g

$f(n) \leq c \cdot g(n)$ for $n \geq n_0$

Ω

Big-Omega: lower bound

f grows at least as fast as g

$f(n) \geq c \cdot g(n)$ for $n \geq n_0$

Θ

Big-Theta: tight bound

f grows exactly like g

both $O(g)$ and $\Omega(g)$

f(n) = 2n + 2	O(n)?	O(n²)?	Ω(n)?	Θ(n)?
	yes	yes, but loose	yes	yes: tight

Measuring Algorithms Without Running Them

IIT Madras Zanzibar

Warm-up Quiz

Why Not Just Time It?

Counting Steps

Big-O, Big-Omega, Big-Theta

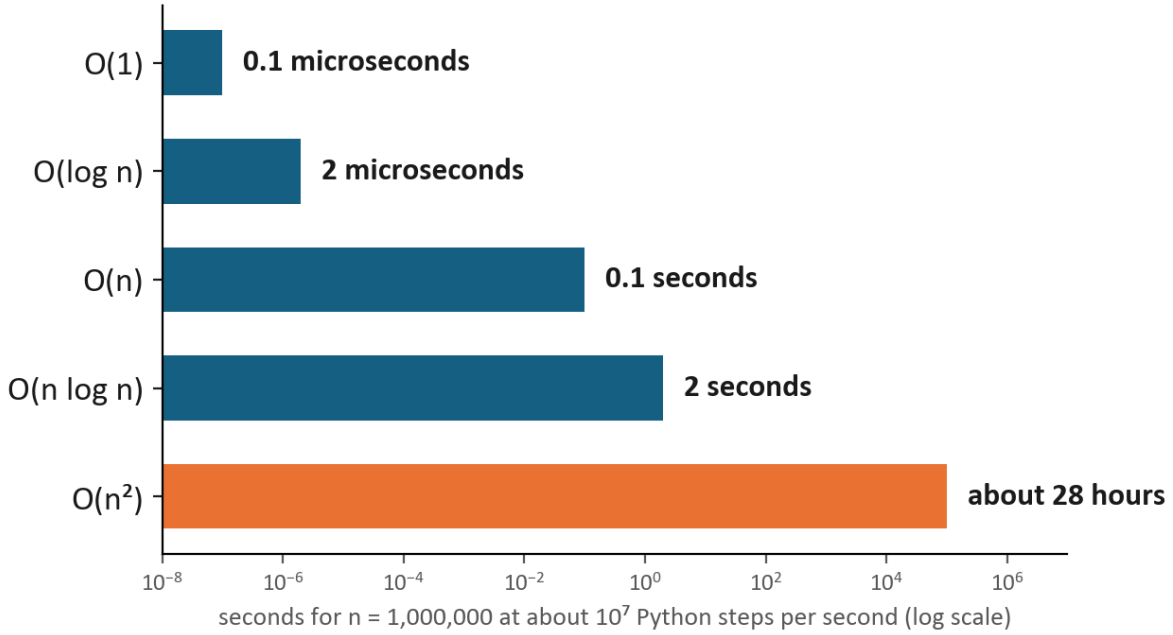
→ **The Growth Classes and Loop Rules**

In-Class Exercise

The Growth Classes You Will Meet All Semester

IIT Madras Zanzibar

Class	Name	Python example
O(1)	constant	a[i], d[key]
O(log n)	logarithmic	binary search
O(n)	linear	for x in a: ..., x in a_list
O(n log n)	linearithmic	sorted(a)
O(n^2)	quadratic	compare every pair



The big-data line: n log n finishes in seconds on a million items; n^2 takes more than a day. Estimates assume about 10^7 Python steps per second.

Three Rules for Loops

IIT Madras Zanzibar

Rule 1: Add one after another

```
for i in range(n): ...
for j in range(n): ...
```

$n + n = O(n)$

Rule 2: Multiply one inside another

```
for i in range(n):
    for j in range(n): ...
```

$n \times n = O(n^2)$

Rule 3: Halve variable halves each time

```
i = n
while i > 1:
    i = i // 2
```

about $\log_2 n = O(\log n)$

Watch out: an inner loop that runs a FIXED number of times (say 10) is a constant: n × 10 is O(n), not $O(n^2)$.

Best, Worst and Average Case

IIT Madras Zanzibar

```
def find(a, target):
    for i in range(len(a)):
        if a[i] == target:
            return i
    return -1
```

1 **Best case**

target is first

1 step: O(1)

2 **Average case**

anywhere, equally likely

about n/2: O(n)

3 **Worst case**

last, or not there

n steps: O(n)

Our default: the worst case. It is a guarantee. A payment system or a hospital database must know the longest it could take, not the luckiest.

Measuring Algorithms Without Running Them

IIT Madras Zanzibar

- Warm-up Quiz
- Why Not Just Time It?
- Counting Steps
- Big-O, Big-Omega, Big-Theta
- The Growth Classes and Loop Rules
- → **In-Class Exercise**

In-Class Exercise (5 minutes, in pairs)

IIT Madras Zanzibar

IN-CLASS TASK: give the tightest Big-O of each snippet, and the rule you used

1

```
for i in range(n):
  print(i)
for j in range(n):
  print(j)
```

2

```
for i in range(n):
  for j in range(n):
    for k in range(10):
      total += 1
```

3

```
i = n
while i > 1:
  i = i // 2
```

4

```
for i in range(n):
  for j in range(i):
    total += 1
```

5 Prove **$5n + 3 = O(n)$**: find constants c and n_o, and show the inequality holds.

Hint: for 4, the inner loop runs 0, 1, 2, ..., n−1 times. What is that sum?

Solution

IIT Madras Zanzibar

1	`1: O(n)`	Rule 1, add: n + n = 2n
2	`2: O(n²)`	Rule 2, multiply: n × n × 10; the 10 is a constant
3	`3: O(log n)`	Rule 3, halve: about $\log_2 n$ iterations
4	`4: O(n²)`	0 + 1 + ... + (n−1) = n(n−1)/2, the same count as program A
5	`5: c = 8, n₀ = 1`	for n ≥ 1, 3 ≤ 3n, so 5n + 3 ≤ 5n + 3n = 8n

Quiz 1 tip: show the rule or the inequality. The reasoning earns the marks.

Summary and Exam Checkpoints

IIT Madras Zanzibar

Count steps

T(n) in the RAM model, not seconds on a stopwatch

Dominant term

drop lower-order terms and constants

O, Ω, Θ

upper, lower, tight; prove O with c and n_0

Loop rules

add, multiply, halve

Next

Thu 8 Oct, 10:00: Foundations: data, data structures and algorithms

Thu 15 Oct, 2:00: Lab 1, our first lab (Week 2): setup and Python warm-up

Read: the rest of Canning, Broder & Lafore, Chapter 2

References and Further Reading

Canning, J., Broder, A. & Lafore, R. (2022). Data Structures & Algorithms in Python. Addison-Wesley. Chapter 2, Big O Notation.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2022). Introduction to Algorithms, 4th ed. MIT Press. Chapter 3.

Skiena, S. S. (2020). The Algorithm Design Manual, 3rd ed. Springer. Chapter 2, Algorithm Analysis.

Big-O Cheat Sheet: bigocheatsheet.com · VisuAlgo: visualgo.net